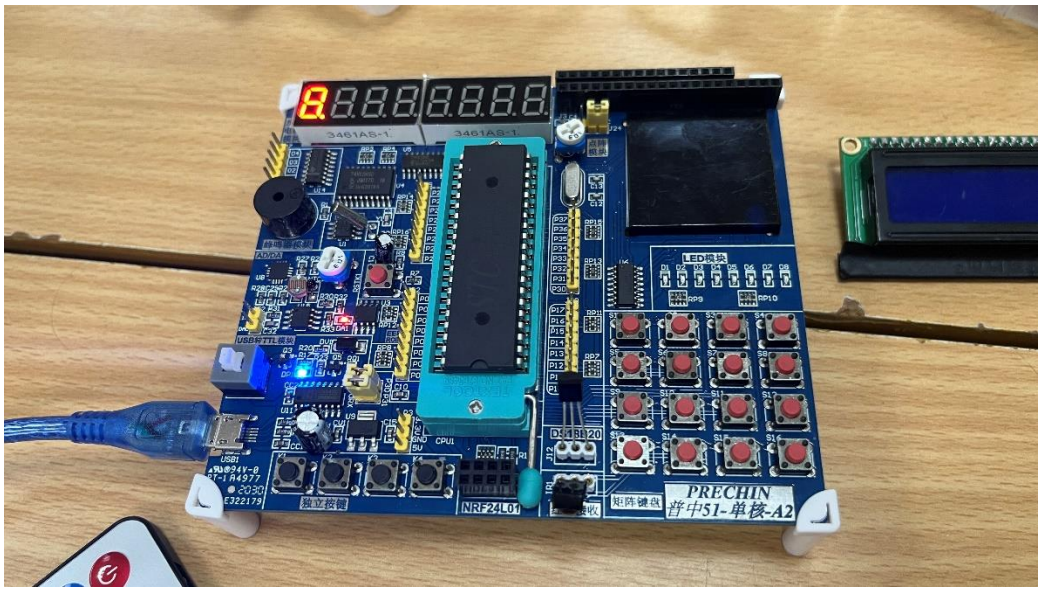
Lab 1 Report

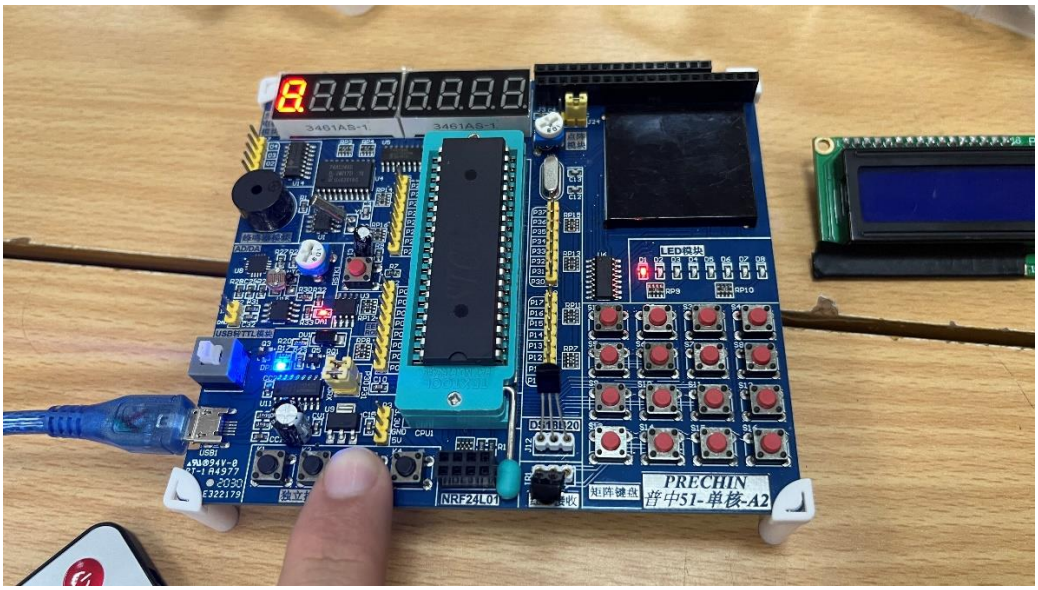
Nguyễn Tiểu Phương 20210692

Ex 3.1

The code provides a simple external interrupt handler. When user pressed the button K3, the board detects the event and lights up the LED D1 accordingly.



Initially, when no button pressing occurs, the LED is off.



When we press the button K3, the LED D1 lights up.

Source Code Explanation

```c
/* Sample: Interrupt handling */

#include <mcs51/8052.h>

#define LED_1    P2_0
#define EX0_PIN  P3_2

void main(void)
{
    //initialize interrupt
    IT0 = 1;
    EX0 = 1;
    EA  = 1;
    EX0_PIN = 1;

    while(1);

}

void ISR0() __interrupt IE0_VECTOR
{
    LED_1 = !LED_1;
}

```

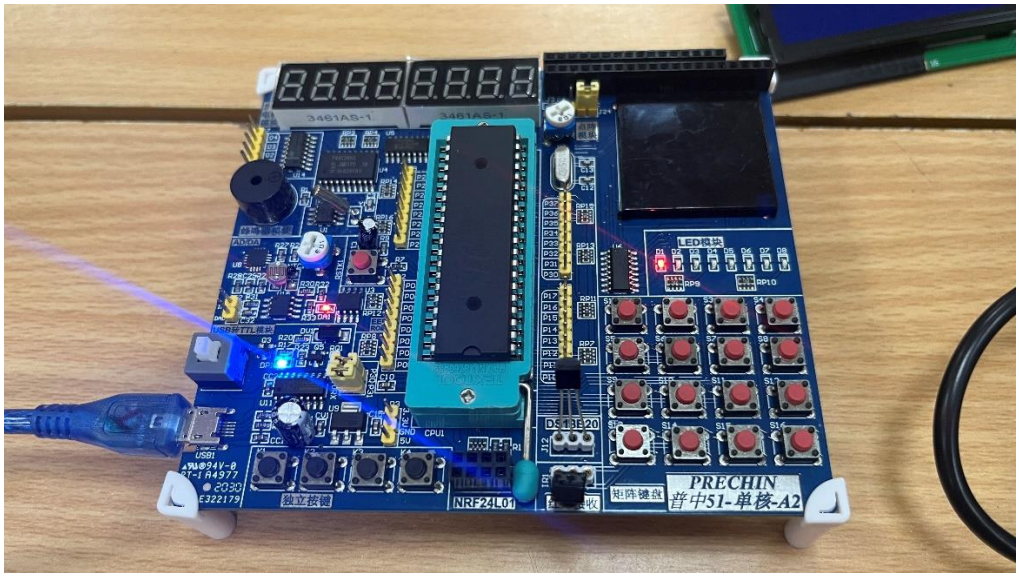
Two macros `LED_1` and `EX0_PIN` is set up to be the controlling ports of the LED D1 and the button K3, respectively.

In the `main()` function, we initialize the interrupt by setting the interrupt signals needed to 1, including IT0, EX0, EA. We also set the button value to 1, indicating the not-pressed status.

In `ISR0()`, we handle the interrupt by setting the LED to its negated value – which means, if the LED is off, it will be turned on.

Ex 3.2

This program ultilizes the timer to make the LEDs blink in a specified amount of time.



The LED D1 will blink continuously.

Source Code Explanation

```
#include <mcs51/8051.h>

#define TH0_50ms        0x4C
#define TL0_50ms        0x00
#define LED_1           P2_0
#define LED7SEG_DATA    P0

int count;

void main(void)
{
    count = 0;
    LED7SEG_DATA = 0x00;

    TMOD = 0x01;    //timer 0 mode 1
    TH0 = TH0_50ms;
    TL0 = TL0_50ms;

    IE  = 0b10000010;
    TF0 = 0;
    TR0 = 1;

    while(1);
}

void TIMER0_ISR() __interrupt TF0_VECTOR
{
    TH0 = TH0_50ms;
    TL0 = TL0_50ms;

    count++;
    if (count == 20)
    {
        count = 0;
        LED_1 = !LED_1;
    }
}

```

We need to set TH0 and TL0 to the value 4Ch and 00h to obtain a delay of 50ms.

The macros **LED_1** and **LED7SEG_DATA** are assigned to the corresponding controlling ports.

In the **`main()`** function, we

- use the Timer 0 of the two timers;
- set the 7-seg LED to 0 (to prevent them from lighting up);
- using the LED D1 (controlled by P2_0)
- set TH0 and TL0 to their respective macros we've defined earlier.

The interrupt enable IE is set to 0b10000010 which means:

- EA = 1 (needed for all interrupt)
- ET0 = 1 (to enable interrupt for Timer 0)

All other interrupt sources is set to 0 because we won't need them.

In `TIMER0_ISR()`, the interrupt is handled as follows:

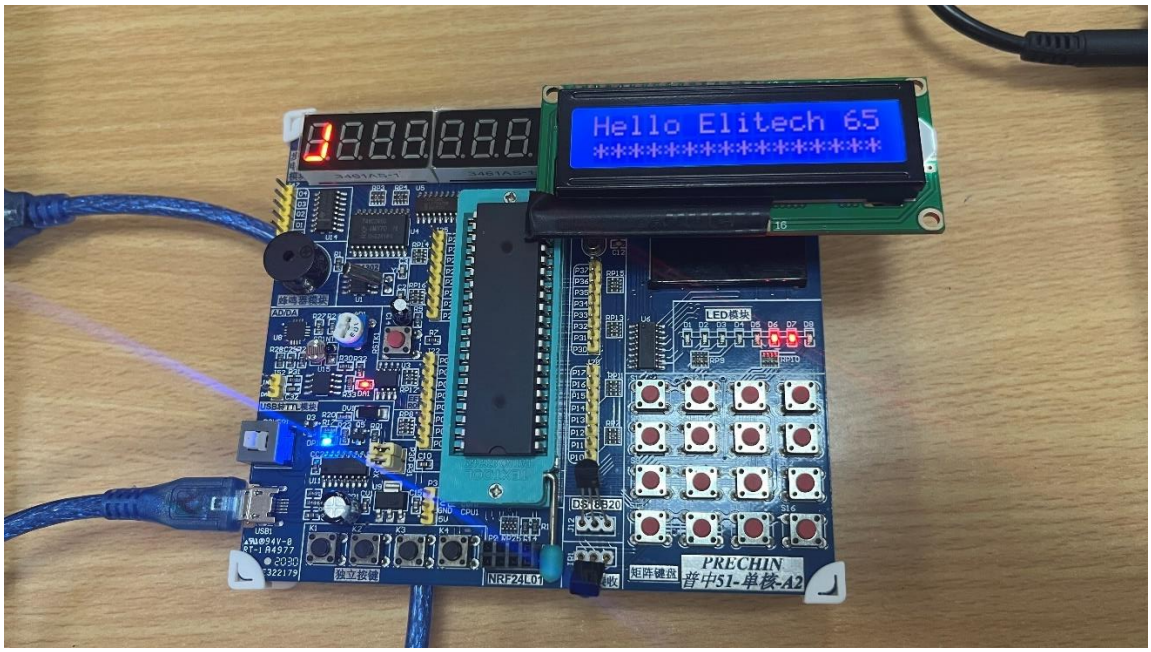
- Increase the count
- If the count reached 20, we reset it and alter the state of the LED D1.

Ex 3.3

This exercise uses the LCD 1602 to display the text:

```
"Hello Elitech 65"
"****************"
```

Additionally, we modified the original code to make the text run all the way left, then all the way right.



The LCD 1602 is displaying the text.

Source Code Explanation

The source code consists of 3 files:

1. A C header file **`lcd1602.h`**, contain all the definitions and function templates we'll be using;
2. **`Lcd1602.c`**, contain the implementation for each of these functions
3. **`Lcd1602demo.c`**, contain the **`main()`** function that we'll run.

```c
#ifndef LCD1602_H_INCLUDED
#define LCD1602_H_INCLUDED

//LCD in 8 bit interface mode
#define LCD_DATA    P0
#define LCD_EN          P2_7
#define LCD_RS          P2_6
#define LCD_RW          P2_5

void LCD_init();
void Wait_For_LCD();
void LCD_Send_Command(unsigned char x);
void LCD_Write_One_Char(unsigned char c);
void LCD_Write_String(unsigned char *s);
void LCD_GotoXY(char row, char col);
void Delay_ms(int interval);
void Delay_us(int interval);

#endif // LCD1602_H_INCLUDED
```

The header file.

```c
#include <mcs51/8052.h>
#include "lcd1602.h"
#include "string.h"

void main()
{
    int i;
    LCD_init();
    LCD_Write_String("Hello Elitech 65");
    LCD_Send_Command(0xC0); //Chuyen con tro xuong dong thu 2
    LCD_Write_String("**********");
    LCD_Send_Command(0xE); //Chuyen con tro xuong dong thu 2
    while(1){
        //---------------
        //your code here
        //---------------
    }
}
```

The demo file.

We will mainly discuss the meaning of functions in the **lcd1602.c** file.

```c
#include <mcs51/8052.h>
#include "lcd1602.h"
#include "string.h"

//Ham gui mot lenh xuong LCD
void LCD_Send_Command(unsigned char x)
{
    LCD_DATA=x;
    LCD_RS=0; //Chon thanh ghi lenh
    LCD_RW=0; //De ghi du lieu
    LCD_EN=1;
    Delay_us(100);
    LCD_EN=0;
    Wait_For_LCD(); //Doi cho LCD san sang
    LCD_EN=1;
}
//Ham kiem tra va cho den khi LCD san sang
void Wait_For_LCD()
{
    Delay_us(100);
}
void LCD_init()
{
    LCD_Send_Command(0x38); //Chon che do 8 bit, 2 hang cho LCD
    LCD_Send_Command(0x0E); //Bat hien thi, nhap nhay con tro
    LCD_Send_Command(0x01); //Xoa man hinh
    LCD_Send_Command(0x80); //Ve dau dong
}
```

`LCD_Send_Command()`

This function sends a command byte `(x)` to the LCD.

- It sets the `LCD_DATA` pin to the command value.
- It clears the `LCD_RS` (Register Select) bit to indicate a command is being sent.
- It clears the `LCD_RW` (Read/Write) bit to indicate a write operation.
- It pulses the `LCD_EN` (Enable) pin to initiate the data transfer.
- It includes a small delay `(Delay_us(100))` to allow the LCD to process the command.
- It calls `Wait_For_LCD()` to ensure the LCD is ready for further communication.

`Wait_For_LCD()`

This function simply call the delay function.

`LCD_init()`

This function initiates the LCD display by assigning the corresponding command to the `LCD_Send_Command()` function.

- It sends the command `0x38` to configure the LCD in 8-bit mode with two lines.
- It sends the command `0x0E` to enable the display, turn on the cursor, and make it blink.
- It sends the command `0x01` to clear the display.
- It sends the command `0x80` to move the cursor to the home position (first line, first column).

```
30  //Ham de LCD hien thi mot ky tu
31  void LCD_Write_One_Char(unsigned char c)
32  {
33      LCD_DATA=c; //Dua du lieu vao thanh ghi
34      LCD_RS=1; //Chon thanh ghi du lieu
35      LCD_RW=0;
36      LCD_EN=1;
37      Delay_us(10);
38      LCD_EN=0;
39      Wait_For_LCD();
40      LCD_EN=1;
41  }
42  //Ham de LCD hien thi mot xau
43  void LCD_Write_String(unsigned char *s)
44  {
45      unsigned char length;
46      length=strlen(s); //Lay do dai xau
47      while(length!=0)
48      {
49          LCD_Write_One_Char(*s); //Ghi ra LCD gia tri duoc tro boi con tro
50          s++; //Tang con tro
51          length--;
52      }
53  }
54
55  void LCD_GotoXY(char row, char col)
56  {
57      char i;
58      if (row == 2)
59          LCD_Send_Command(0xC0);      //cursor to fist col in row 2
60      else
61          LCD_Send_Command(0x80);      //cursor to fist col in row 1 (default)
62      for (i = 0; i < col; i++)
63          LCD_Send_Command(0x14);      //cursor to fist col in row 1 (default)
64  }
```

`LCD_Write_One_Char`

This function take in each char of the string, load it into the `LCD_DATA` register.

- It sets the `LCD_RS` bit to indicate data is being sent.
- It clears the `LCD_RW` bit for a write operation.
- It pulses the `LCD_EN` pin to initiate the data transfer.
- It includes a smaller delay `(Delay_us(10))` compared to `LCD_Send_Command` as character writes are generally faster.

`LCD_Write_String`

- This function writes a string `(s)` to the LCD.
- It first gets the string length using strlen from the string.h library.
- It iterates through each character in the string using a while loop.
- Inside the loop, it calls `LCD_Write_One_Char` to write the current character pointed to by the string pointer (*s).
- It then increments the string pointer (s++) to move to the next character in the string.
- The loop continues until the entire string has been written.

`LCD_GotoXY`

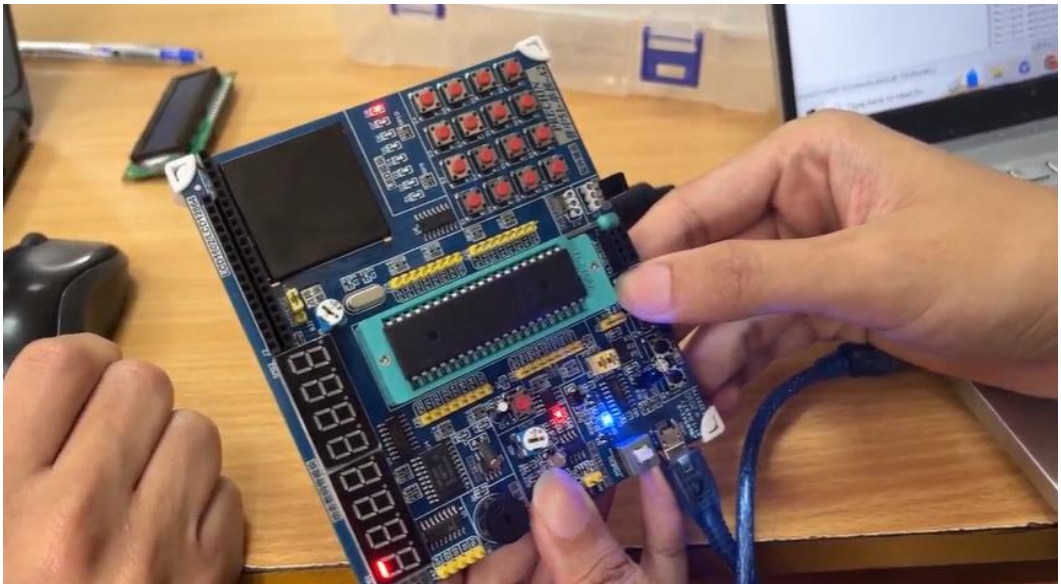
- This function positions the cursor on the LCD at a specific location (row, col).
- It checks the row value and sends the appropriate command (`0xC0` for second row, `0x80` for first row) to move the cursor to the first column of the desired row.
- It then iterates a loop (col times) sending the command `0x14` (shift cursor right) to move the cursor to the desired column position within the same row.

```
66  void Delay_ms(int interval)
67  {
68      int i,j;
69      for(i=0;i<1000;i++)
70      {
71          for(j=0;j<interval;j++);
72      }
73  }
74
75  void Delay_us(int interval)
76  {
77      int j;
78      for(j=0;j<interval;j++);
79  }
```

These two delays are pretty self-explanatory. The first one create a delay in milliseconds (ms), while the second one create a delay in microseconds (us).

Ex 3.4

In this exercise, we program the board so that it detects the event of a double click, which results in the LED D8 being turned on. A valid double click is defined to be two consecutive button press events with time gap of no more than 200ms.



To ensure proper demonstration of the board, a video (rather than a still photo) is needed. **For the imagery of the working circuit, please refer to this video.**

Source Code Completion

We complete the source code by writing the following inside the infinite loop:

```
11  void main(void)
12  {
13      count = 0;
14      key_count = 0;
15      LED7SEG_DATA = 0x00;
16
17      TMOD = 0x02;     //timer 0 mode 1
18      TH0 = 25;        //auto reload, period ~0.25ms
19      TL0 = 0;
20
21      IT1 = 1;
22      IE  = 0b10000110;
23      TF0 = 0;
24      TR0 = 1;
25
26      while(1)
27      {
28          if (key_count ≥ 2 && key_duration[key_count] ≤ 800) {
29              LED = 0;
30              key_count = 0;
31          }
32          else if (count ≥ 2000) {
33              LED = 1;
34              key_count = 0;
35              count = 0;
36          }
37      }
38  }
```

To detect the event of a double click, we check two conditions:

1. The `key_count` is greater or equal to 2;
2. The duration of the current `key_count` (which is the period in between the current `key_count` and the previous `key_count`) satisfies the time span of 200ms. Since we reuse the code with period 0.25ms; we have 200/0.25 = 800, which is the maximum valid key duration.

For the else clause, we handle when “K4 is not pressed again in 500ms” condition by checking whether count exceeds 2000 (since 2000 * 0.25 = 500ms).

Additional notes:

- `key_count` is checked whether it is greater or equal to 2, rather than just equal 2 (to recognize a double click), because we need to account for the limit of the hardware, the jammed key phenomenon (Hiện tượng nảy phím). Here, we resolve the problem using logic of our software.
- `key_count` should be reset to 0 upon each condition check to ensure that the code is logically correct and NOT induce the warning from the optimizer: *Unreachable code.*

Source Code Explanation

```
32  void TIMER0_ISR() __interrupt TF0_VECTOR
33  {
34      count++;
35  }
36
37  void EX1_ISR() __interrupt IE1_VECTOR
38  {
39      EX1 = 0;
40      key_count++;
41      if (key_count > 3)
42          key_count = 3;
43      key_duration[key_count] = count;
44      count = 0;
45      EX1 = 1;
46  }
```

The `TIMER0_ISR()` purely used for counting.

`EX1_ISR()` is the routine where we handle the interrupt.

Initially, we set EX1 to 0.

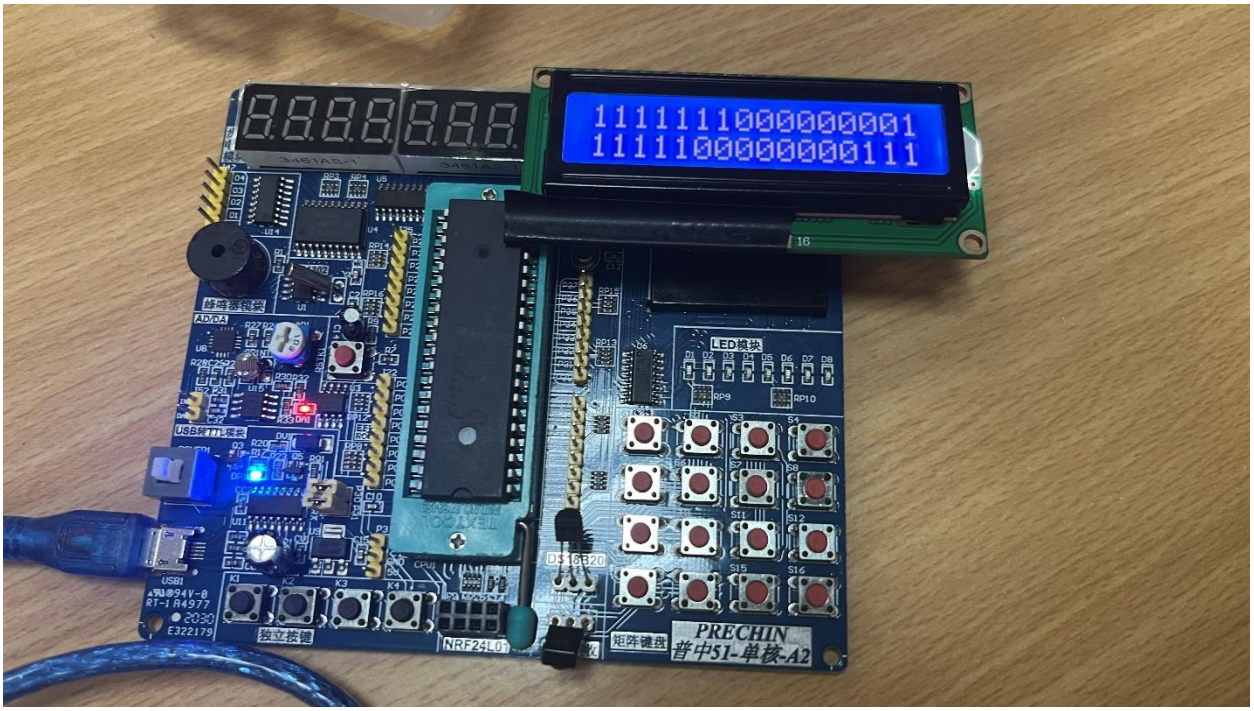
Each time the program enters this routine, it means that the key is pressed one more time – so we increase `key_count`.

If `key_count` is more than 3, we only regard them as 3 at most. Multiple clicks should be regarded as a single double click. From a design perspective, this is for simplicity and jittery avoidance.

We set the `key_duration` of the corresponding `key_count` to `count`, reset it to 0, and reset EX1 to 1. These are the usual reset steps for the next button press event.

Ex 4

The goal of this exercise is similar to that of the exercise 3.4, but this time, the event we are handling is the emitted infared recognized by the IR Receiver Modules HS0038 when we press buttons of the remote controller.



When the board displays the right frame values.
In class, due to the instability of the board as well as some logic errors of the code, the board did not achieve 100% correctness yet.

Please refer to this video for more of the working circuit.

Source Code Completion

Since the functionalities are somewhat similar, we will re-use some source code from the previous exercises.

Algorithm Explanation

Since a NEC frame contain 32 bits, we ultilize all of the LCD display slots.

We declare **`cmd1`** and **`cmd2`** to be arrays that hold the first and the second half of the frame value, respectively.

To make the board behavior easier to observe at each step, we set the value of P2 to turn the LED on or off.

If after a long time (count > 5000) and no signal received, we turn off the LED and reset **`count`**.

Otherwise, when `key_count` >= 1 that means a start signal has been recognized, and we start reading each of the following pulses. While this logic does not handle the start of the frame in a theoretically correct way, we haven't come up with an alternative in class' time.

Inside the two for loop, we iterate through the pulses and assign value 0 or 1 to the command array based on their duration (`dur` variable). `j` is for the index of each char inside the command array.

The `while (key_count < 2)` loop is to wait until we received at least 2 more pulses.

Note: After throughout testing, we figured that the number for the range of recognizing 0 or 1 can be "tuned" to maximize the probability of "catching" the correct value, but this is not guaranteed.

For example, the code below will assign a '0' if dur is less than 7 (since a '0' requires theoretically 5 counts = 1.25ms); while it will assign a '1' if dur is in the range (7, 30) (since a '1' is equivalent to 9 counts = 2.25ms).

We can change the value '7' and '30' to any of our preferences, as long as the range for each bit is correct.

Upon successful decoding, we print the frame to the LCD display.

Full Code

```
#include <mcs51/8051.h>

#define LED             P2_7
#define LED7SEG_DATA    P0
#define BUTTON          P3_3
#define IR1             P3_2

int count;
int key_duration[4];
unsigned char key_count;

void main(void)
{
```

```
    count = 0;
    key_count = 0;
    LED7SEG_DATA = 0x00;

    TMOD = 0x02;    //timer 0 mode 1
    TH0 = 25;       //auto reload, period ~0.25ms
    TL0 = 0;

    IT1 = 1;
    IE  = 0b10000110;
    TF0 = 0;
    TR0 = 1;

    char cmd1[20];
    char cmd2[20];
    while(1)
    {
        if (count > 5000) {
            P2 = 0xff;
            count = 0;
        }

        if (key_count >= 1) {
            key_count = 0;
            P2_1 = 0;
            P2_7 = 0;
            int j = 0;
            for (int i = 0; i < 16; i++) {
                while (key_count < 2);
                dur = key_duration[2];
                key_count = 1;
                if (dur < 7) {
                    cmd1[j] = '0';
                } else if (dur < 30) {
                    cmd1[j] = '1';
                }
                j++;
            }
            cmd1[j] = '\0';

            j = 0;
            for (int i = 16; i < 32; i++) {
                while (key_count < 2);
                dur = key_duration[2];
                key_count = 1;
```

```
                if (dur < 7) {
                    cmd2[j] = '0';
                } else if (dur < 30) {
                    cmd2[j] = '1';
                }
                j++;
            }
            cmd2[j] = '\0';

            LCD_init();
            LCD_Write_String(cmd1);
            LCD_Send_Command(0xC0);
            LCD_Write_String(cmd2);
            LCD_Send_Command(0xE);
        }
        key_count = 0;
        count = 0;
    }

}


void TIMER0_ISR() __interrupt TF0_VECTOR
{
    count++;
}

void EX1_ISR() __interrupt IE1_VECTOR
{
    EX1 = 0;
    key_count++;
    if (key_count > 3)
        key_count = 3;
    key_duration[key_count] = count;
    count = 0;
    EX1 = 1;
}
```

THE END